Assignment 4 solutions

1a) Table: Since we need to keep track of which pieces are used, use a 2D table with rod size and piece size.
$T[0 \ldots n, 0 \ldots n]$ where $T[i,j]$ is the max value possible when cutting a rod of length i, using piece sizes at most j.

Initialisation: $T[0,j] = 0 \quad \forall j = 0 \ldots n$ (no value, rod has no length)
$T[i,0] = -\text{inf} \quad \forall i = 1 \ldots n$ (set to min possible cost, since no pieces can be cut).

Relationship: Consider $T[i,j]$:

- if $j > i$, piece j can't be used. $\therefore T[i,j] = T[i,j-1]$
- if $j \le i$, take best price out of using piece j or not:

$$\underbrace{T[i-j}_{\text{use piece } j}, \underbrace{j-1]}_{\text{don't use again}} + p[j] \quad \text{vs.} \quad \underbrace{T[i,j-1]}_{\text{don't use } j}$$

Return value: $T[n,n]$ considers entire rod, using all possible piece sizes.

Pseudo Code:

```
for j = 0 to n
        T[0,j] = 0
for i = 1 to n
        T[i,0] = -inf.     // important to set this to an "impossible" value.
for i = 1 to n
        for j = 1 to n
                if j > i
                        T[i,j] = T[i,j-1]
                else
                        T[i,j] = max{ T[i-j, j-1] + p[j], T[i,j-1] }.
Return T[n,n].
```

Runtime: Each table entry is filled in in constant time. For a table of size $\Theta(n^2)$, the runtime is $\therefore \Theta(n^2)$

Q2) Table: $T[1 \ldots m, 1 \ldots n, 1 \ldots p]$ where
$T[i,j,k]$ = max chance of survival from $(1,1,1)$ to (i,j,k)

Initialisation: Let a = # of assassins at $(1,1,1)$.
$\therefore$ survival rate is $(0.9)^a$. Set $T[1,1,1] = (0.9)^a$.

Relationship: Each location (i,j,k) has at most three possible previous locations.
Out of those possible previous locations, we take the maximum of

the corresponding table entries. Then we multiply this maximum by the survival rate at (i,j,k), which is $0.9^{A[i,j,k]}$.

Pseudo-Code:

```
a = A[1,1,1]
T[1,1,1] = 0.9^a
for i = 1 to m
    for j = 1 to n
        for k = 1 to p
            if (i, j, k) ≠ (1,1,1)          // fill in all entries except (1,1,1)
                prevx = prevy = prevz = 0   // possible previous routes. Set to 0.
                if i > 1                    // prev. route along x coord. exists.
                    prev-x = T[i-1, j, k]
                if j > 1                    // prev. route along y coord exists
                    prev-y = T[i, j-1, k]
                if k > 1                    // prev route along z coord exists.
                    prev-z = T[i, j, k-1]
                a = A[i,j,k]
                best = max { prevx, prevy, prevz }   // best prev. step.
                T[i,j,k] = 0.9^a x best
Return T[m,n,p]
```

Runtime: Each table entry is filled in in constant time. ∴ runtime is $O(m \cdot n \cdot p)$.

(b) The solution below traces the max route from (m,n,p) to $(1,1,1)$, printing route in reverse.

```
i = m, j = n, k = p.
while (i,j,k) ≠ (1,1,1)        // search from (m,n,p) → (1,1,1)
    Print (i,j,k)
    prevx = prevy = prevz = 0
    if i > 1
        prevx = T[i-1, j, k]
    if j > 1
        prevy = T[i, j-1, k]
    if k > 1
        prevz = T[i, j, k-1]
    if prevx > prevy and prevx > prevz      // prevx is max
        i = i-1
    else if prevy > prevx and prevy > prevz // prevy is max
        j = j-1
    else z = z-1                            // prevz is max.
```

Runtime: $O(m+n+p)$

Q3 a) Same as LIS problem from week 9, where we find an increasing sequence by age and height.

Step 1) Sort boxes by length. Runtime: $O(n^2)$

Run Insertion Sort on $L[1 \ldots n]$ where a swap is carried out on all three arrays! (Pseudo Code not necessary)

```
for i = 2 to n
    for j = i downto 2
        if L[j] < L[j-1]
            Swap L[j] and L[j-1]
            Swap W[j] and W[j-1]
            Swap H[j] and H[j-1].
        else break.
```

Step 2) Run Longest Increasing Subseq. using max of tower height!

Table: $T[i]$: max height of tower using boxes $1, 2, \ldots i$ where box i is selected as Bottom Box.

$Prev[i]$: index of box placed on top of box i.

Initialisation: $T[1] = H[1]$ (Height of a tower with only 1 box)

$Prev[1 \ldots n] = 0$. Set all boxes as potential "top" boxes.

Relationship: Consider box K, and loop through all possible boxes $i = 1$ to $K-1$ with $L[i] \leq L[K]$ and $W[i] \leq W[K]$. Out of all those possible boxes, pick the one that has the highest tower height $T[i]$, and add to it height $H[K]$

Pseudo Code:

```
for K = 2 to n
    max = H[K]    // tower height with only box K.
    for i = 1 to K-1    // possible boxes on top of box K.
        if W[i] ≤ W[K]
            if T[i] + H[K] > max.
                max = T[i] + H[K]
                Prev[K] = i
    T[K] = max.
Return max { T[1], ... T[n] }.
```

Runtime: Same structure as L.I.S, with runtime of $O(n^2)$.

(b) Let K = index of maximum in T[1...n]. // index of bottom box.
While Prev[K] ≠ 0 // continue until find top box
    Print L[K] × W[K] × H[K]
    K = Prev[K] // index of box on top of box K.

Q4 Table: Define table P[0...m, 0...n, 0...p] where P[i, j, K] represents the chance of survival for each species, assuming initial population of i dragons, j boars, K coywolves.

Initialisation: P[0,0,0] = [0,0,0]
P[i,0,0] = [1,0,0] for i = 1... m
P[0,j,0] = [0,1,0] for j = 1...n
P[0,0,K] = [0,0,1] for K = 1...p.

All dragons killed: P[i,j,0] = [0,1,0] for i = 1... m, j = 1...n
All coywolves killed: P[i,0,K] = [1,0,0] for i = 1...m, K = 1...p
All boars killed: P[0,j,K] = [0,0,1] for j = 1...n, K = 1...p.

Relationship:
If a dragon meets a boar, the dragon is killed. This happens with probability $i \cdot j/(ij + jK + iK)$
If a boar meets a wolf, the boar is killed. This happens with probability $jK/(ij + jK + iK)$
If a wolf meets a dragon, the wolf is killed. This happens with probability $i \cdot K/(ij + jK + iK)$

Return value: P[m,n,p] represents the survival rate assuming initial populations of m,n,p for each of dragon, boar, and wolf populations.

Pseudo code:

```
P[0,0,0] = [0,0,0].
for i = 1 to m
        P[i, 0, 0] = [1,0,0]
        for j = 1 to n
                P[i, j, 0] = [0,1,0].   // all dragons are killed by boars.
for j = 1 to n
        P[0, j, 0] = [0,1,0]
        for K = 1 to p
                P[0, j, K] = [0,0,1]   // all boars are killed by wolves
for K = 1 to p
        P[0,0,K] = [0,0,1]
        for i = 1 to m
                P[i, 0, K] = [1,0,0].   // all wolves are killed by dragons.
for i = 1 to m
    for j = 1 to n
        for K = 1 to p.
                interspecies = i·j + j·K + i·K   // total # of possible meetings
                                                 between species.
```

dead-d = ij / interspecies. // chance of dead dragon
dead-b = $j \cdot K$ / interspecies. // chance of dead boar
dead-c = $i \cdot K$ / interspecies. // chance of dead coywolf.

$$P[i, j, K] = \text{dead-d} \times P[i-1, j, K] + \text{dead-b} \times P[i, j-1, K] + \text{dead-c} \times P[i, j, K-1]$$

Return $P[m, n, p]$.

Runtime: Each cell is filled in in constant time. ∴ overall runtime is $O(m \cdot n \cdot p)$

Q5) Table: Let $P[i, j, b]$ be defined for $i = 1 \ldots n$, $j = 0 \ldots K$ and b = "own" or b = "free"
where:

- entry $P[i, j, \text{own}]$ is the max profit obtainable from days $1 \ldots i$ using at most j transactions, where on day i you own a commodity.
- entry $P[i, j, \text{free}]$ is the max profit obtainable from days $1 \ldots i$ using at most j transactions, where on day i we don't own a commodity.

Initialisation:

$P[i, 0, \text{own}] = P[i, 0, \text{free}] = 0$, for all $i = 1 \ldots n$, since no transactions are possible.
$P[1, j, \text{own}] = -c[1]$ since on day 1 we could by commodity 1 at price of $c[1]$
$P[1, j, \text{free}] = 0$ since we could make no purchase on day 1.

Relationship:

1) If we don't own on day i, it's either because we didn't own the day before OR we owned the day before but chose to sell on day i.

$$P[i, j, \text{free}] = \max\{\underbrace{P[i-1, j, \text{own}] + p[i]}_{\text{owned prev day and sold on day } i.}, \underbrace{P[i-1, j, \text{free}]}_{\text{didn't own prev. day}}\}$$

2) If we own on day i, it's either because we owned the previous day, OR we didn't own the previous day and made purchase on day i.

$$P[i, j, \text{own}] = \max\{P[i-1, \underset{\substack{\uparrow \\ \#\text{transactions decreased}}}{j-1}, \text{free}] \underset{\substack{\uparrow \\ \text{purchased on day } i}}{-\, p[i]}, \underbrace{P[i-1, j, \text{own}]}_{\text{owned on day } i-1.}\}.$$

Pseudocode.

```
for i = 1 to n
        P[i, 0, own] = P[i, 0, free] = 0
for j = 1 to K
        P[1, j, own] = - c[1]
        P[1, j, free] = 0
```

for $i = 2$ to n
    for $j = 1$ to k
        $P[i, j, \text{free}] = \max\{P[i-1, j, \text{own}] + p[i],\ P[i-1, j, \text{free}]\}$
        $P[i, j, \text{own}] = \max\{P[i-1, j-1, \text{free}] - p[i],\ P[i-1, j, \text{own}]\}$
Return $P[n, K, \text{free}]$

Return value: Considering max of K transactions, assuming we don't need to own anything on last day, return $P[n, K, \text{free}]$.

Runtime: The body of the for loop runs in constant time. Therefore the overall runtime is $O(n \cdot K)$.

Ex Practice filling in this table! You should get a max profit of 16 for ≤ 5 transactions.

free, own

i		j: 0	1	2	3	4	5
1	3	0,0	0,-3	0,-3	0,-3	0,-3	0,-3
2	2	0,0					
3	5	0,0					
4	4	0,0					
5	7	0,0					
6	1	0,0					
7	5	0,0					
8	6	0,0					
9	4	0,0					
10	9	0,0					